IEEE ComSoc | IEEE COMPUTER SOCIETY | IEEE Signal Processing Society | VTS | IEEE Transactions on Machine Learning in Communications and Networking

Received 6 June 2023; revised 29 December 2023 and 27 March 2024; accepted 4 April 2024.
Date of publication 9 April 2024; date of current version 16 April 2024.

The associate editor coordinating the review of this article and approving it for publication was H. Ko.

Digital Object Identifier 10.1109/TMLCN.2024.3386647

Fast Context Adaptation in Cost-Aware Continual Learning

SEYYIDAHMED LAHMER (Student Member, IEEE), FEDERICO MASON (Member, IEEE), FEDERICO CHIARIOTTI (Member, IEEE), AND ANDREA ZANELLA (Senior Member, IEEE)

Department of Information Engineering, University of Padua, 35131 Padua, Italy

CORRESPONDING AUTHOR: F. CHIARIOTTI (chiariot@dei.unipd.it)

This work was supported in part by the European Union (EU) H2020 Marie Skłodowska-Curie Action (MSCA) Innovative Training Network (ITN) Project Greenedge under Grant 953775; in part by EU through Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, as a part of the REDIAL Young Researchers under Grant SoE0000009; and in part by the partnership on "Telecommunications of the Future" under Grant PE0000001-program "RESTART."

ABSTRACT In the past few years, Deep Reinforcement Learning (DRL) has become a valuable solution to automatically learn efficient resource management strategies in complex networks with time-varying statistics. However, the increased complexity of 5G and Beyond networks requires correspondingly more complex learning agents and the learning process itself might end up competing with users for communication and computational resources. This creates friction: on the one hand, the learning process needs resources to quickly converge to an *effective* strategy; on the other hand, the learning process needs to be *efficient*, i.e., take as few resources as possible from the user's data plane, so as not to throttle users' Quality of Service (QoS). In this paper, we investigate this trade-off, which we refer to as *cost of learning*, and propose a dynamic strategy to balance the resources assigned to the data plane and those reserved for learning. With the proposed approach, a learning agent can quickly converge to an efficient resource allocation strategy and adapt to changes in the environment as for the Continual Learning (CL) paradigm, while minimizing the impact on the users' QoS. Simulation results show that the proposed method outperforms static allocation methods with minimal learning overhead, almost reaching the performance of an ideal out-of-band CL solution.

INDEX TERMS Resource allocation, reinforcement learning, cost of learning, continual learning, meta-learning, mobile edge computing.

I. INTRODUCTION

THE role of Artificial Intelligence (AI) in communication networks has become more and more central with the transition from 4G to 5G, and learning is at the core of the 6G standardization process [1]. Mobile networks are evolving beyond rigid entities that users must adapt to, shifting towards customizable services that dynamically respond to users' needs [2]. The Network Slicing (NS) paradigm supports this approach by enabling the definition of multiple logical network *slices* overlaying the same physical infrastructure [3], with each slice devoted to a specific class of service. This allows applications with very different requirements to coexist and share resources. However, managing NS, as well as other advanced application scenarios, requires a dynamic allocation of both transmission and computational resources to users, according to their QoS targets, in a fast-paced scenario [4], which is expected to become even more complex and challenging with 6G. Hence, resource allocation schemes must be able to detect changes in the environmental conditions, as well as in users' and services' requirements. The allocation policy should then adapt accordingly to ensure efficient utilization of the available resources.

Hand-designed resource allocation strategies may not be up to this challenge, and growing attention has been dedicated to machine-learning approaches. In particular, DRL is considered a promising framework for deriving adaptable and robust strategies for network orchestration [4] and resource allocation [5]. DRL's *effectiveness* in dealing with complex scenarios is indeed well-established: with proper training, the DRL agents can find foresighted policies aiming for long-term objectives [6], significantly improving network performance. Such promising results, however, have been typically obtained in stationary environments: if this assumption is not satisfied, the performance of pre-trained DRL

agents may dramatically decrease when the network dynamic shifts away from the training environment, while dynamic environments require not only an effective performance after convergence but also quick reactions to changes, i.e., a high training *efficiency*.

Approaches based on the CL paradigm [7] are designed to deal with non-stationary systems. CL enables the adaptation of a learning agent to a series of subsequent tasks that, in a network scenario, may represent different network configurations. However, combining CL and DRL for managing network resources in non-stationary scenarios has a non-negligible cost in terms of energy, computation, and communication resources [8], due to the complexity of the training process.

The existing literature has extensively explored the trade-off between the *effectiveness* of learned strategies and their *efficiency*, considering the associated costs in terms of computational, communication, and energy resources across various scenarios. On the other hand, to the best of our knowledge, no prior research has specifically delved into this trade-off in case the learning algorithms and the users *share the same pool of resources.* In this context, any resource used by the learning process is subtracted from the *data plane*, i.e., the part of the system that is responsible for transmitting, processing, and forwarding user data packets. In such a setting, indeed, supporting CL represents an overhead for the system, which can negatively impact users' QoS: the resources required for a learning agent to rapidly adapt to changes in the environment may, in fact, surpass the performance gains achieved through such adaptation. Any resources allocated to the *learning plane*, i.e., to communicate experience samples to the Cloud for processing, are taken away from the data plane, and CL requires online updates and new experience, as an offline pre-training is not sufficient to adapt to new environment conditions. Therefore, the DRL agents are required to choose wisely when and how much to train, avoiding the use of resources that must be subtracted from the data plane when this strongly affects the users' QoS. We introduce the term *cost of learning* to indicate the impact that the learning process can have on user performance due to the competition for the same resources.

The cost of learning problem is particularly critical considering the ever-larger size of most recent DRL neural networks, and the growing demand for efficient systems, as for the green networking paradigm [9]. We observe that Mobile Edge Computing (MEC) [10] solutions do not solve the problem, but just shift it to the network edge. In fact, while MEC allows computationally expensive tasks (such as the training of DRL algorithms) to be carried out directly in dedicated edge nodes physically close to the data sources, the limited transmission, computational and energetic resources of such nodes still have to be shared between the data and learning planes. Finding a balance between the number of resources to be used for improving the system reactivity to variations and those to be allocated to serve the users may be a very difficult task. This is particularly critical in the case of CL systems, in which agents must constantly adapt to new working conditions. As the very same network resources are also used for the training, a trade-off between the capability of DRL agent to learn new tasks and its performance during the current task arises.

Note that, although at a first glance this problem may remind readers of the well-known exploration-exploitation problem in learning systems, there is a fundamental difference: the exploration-exploitation problem involves finding a balance between exploring new strategies, with the risk of temporarily worse performance, and exploiting the currently learned strategy (that, however, may be globally suboptimal). In this setting, the resources required by the learning process are typically ignored: whichever action the system chooses, the outcome is assumed to be available to the learner, enriching its experience and improving its policy in future steps, without any cost. On the other hand, the trade-off we consider is beyond the standard DRL formulation, and requires an external solution: we consider the resources needed to transfer experience to the learner and use it to update the policy, irrespective of whether it originates from an exploration or exploitation action. From a theoretical perspective, the scenario we look at is hence a Meta Learning (MeL) problem, in which the agent's actions determine the efficiency of the learning data aggregation and processing.

The cost of learning is therefore a fundamental aspect to be considered in modern network design, and recent works have proposed learning-based frameworks that are computation-aware [11]. Despite the high interest of the scientific community in this field, the cost of learning for DRL models is still a relatively unexplored subject in the networking literature, and even the most recent works on resource allocation and NS ignore the true cost of combining DRL and CL in modern networks [12], making the effectiveness of state of the art DRL solutions questionable.

In this work, we design a novel CL framework to address the trade-off between *effectiveness* and *efficiency* in learning-based resource allocation scenarios. We formally define the resource allocation problem, including the cost of learning, and show that our solution achieves performance close to the ideal upper bound obtained by neglecting the cost of learning. The proposed scheme enables the maximization of the training efficiency (i.e., reducing the learning plane overhead) while still achieving effective resource allocation (i.e., the same QoS as the ideal approach that assumes learning does not consume users' resources) in a reasonable time. Although we applied our solution to a networking scenario, the framework can be directly applied to any learning-based allocation problem in which the allocated resources are also required for the agent training, such as MEC job scheduling.

To the best of our knowledge, this is the first work to consider the cost of learning in terms of the trade-off between training efficiency and policy effectiveness. The major contributions of our work are summarized in the following points:

- we define a theoretical model to analyze the trade-off between the *effectiveness* and *efficiency* in learning-based resource allocation scenarios;
- we propose a novel framework to solve the cost of learning problem, allowing the DRL agent to quickly adapt to changes in the environment statistics according to a CL approach;
- we test the proposed framework in a NS use case, in which the learning agent and system users compete for the same network resources;
- we compare the benefits and drawbacks of our approach against a static resource-sharing scheme between data and learning planes and an ideal strategy that considers out-of-band resources for the agent training (or, equivalently, assumes the learning agent does not consume any user-plane resources); our simulation results show that the proposed heuristic performs closely to the ideal (out-of-band) approach, minimizing the impact of learning plane traffic during the training.

A partial and preliminary version of this work was presented at the 2023 IEEE International Conference on Communications [13]. The conference version introduced the concept of cost of learning, but proposed a basic version of the system model and addressed the problem only in a stationary environment. In this manuscript, we extend that work by introducing the CL approach, optimizing the proposed heuristic, providing a much richer set of results, and deepening the discussion and analysis of our observations.

The rest of this paper is organized as follows: first, in Sec. II, we report the most significant related work; we then present the model for optimizing data and learning plane in Sec. III and define our proposed cost of learning-aware solution in Sec. IV. We define the NS use case in Sec. V, while Sec. VI presents the simulation results in that scenario. Finally, Sec. VII concludes the paper and discusses some possible avenues for future work.

II. RELATED WORK

While the latest advances in AI have made it possible to reach stunning performance levels in multiple fields, there is still a large gap between human cognition and AI models in terms of adaptation. Most of the current learning models need to be retrained from scratch every time a new task has to be accomplished, with a high cost in terms of computational power and time. For this purpose, the scientific community has recently leveraged the CL paradigm, which focuses on learning a series of subsequent data, associated with different tasks, without *catastrophically forgetting* past knowledge [14]. Therefore, in CL scenarios the goal is to adapt to a time-varying environment, working on one task at a time and assuming that future information is inaccessible. This model appeals to the resource allocation problem considered in this manuscript, since in realistic networks the type, number, and requirements of the users keep changing over time, making the system non-stationary (though stationarity can be assumed during the *coherence intervals*, i.e., the time periods during which the main system parameters do not change).

A baseline CL solution may involve a pre-trained model that is iteratively adapted to new tasks (or to changes in the environment), e.g., taking advantage of curriculum learning, as done in [15]. Replay-based methods form a more recent class of CL algorithms, which store past experience in memory or exploit a generative model to reproduce it, using this information as model input while training on new tasks [16], [17]. Regularization-based methods, which introduce a penalty term in the model's loss function with the goal of avoiding performance degradation in past tasks [18], [19], form another class of solutions. An extension of the aforementioned class is proposed in [20], where the authors estimate the importance of each learned parameter and prevent the modifications of such parameters that most affect performance in past tasks. Finally, architecture-based methods define an additional branch of the model for each task, freezing the previously learned parameters when training the model on new scenarios [21], [22]. An example is provided in [23], where the authors developed a two-block model: the first is retrained every time a new task arises, while the latter distills the knowledge acquired for future reuse.

From a different perspective, CL algorithms aim at defining a strategy to detect the best settings for training a learning model in new scenarios. This concept falls within the MeL paradigm, which focuses on convergence speed and stability [24]. MeL methods differ in the output of the learning optimization, which may include the weight initialization strategy [25], the optimizer algorithm [26], the loss function [27], the dimensions of the learning architecture [28], and other hyper-parameters.

The methodology used for the MeL optimization task may be based on gradient descent [29], evolutionary algorithms [30], or learning-based approaches. For instance, the authors of [31] develop a CL model in which a DRL agent has to define the optimal settings of the task-specific block, balancing between validation accuracy and model complexity. Besides, MeL methods differ in terms of optimization goals, which may either be fully based on the model performance on a validation set or consider more specific aspects, such as the adaptability of the solution to multiple tasks [32], the greater importance of fast adaptation than asymptotic performance [33], and the difference between online and offline learning scenarios [34].

In particular, the use of MeL methods for optimizing DRL models is a relatively new field. In this scenario, the combination of CL and MeL avoids the need for a centralized agent that can handle each possible state-action pair and enables the definition of multiple and more straightforward policies. The authors of [35] show the benefits of MeL in a real-world scenario, analyzing a DRL robotic system that exploits a recurrent module to preserve past knowledge and speed up the training process. Interestingly, when applying MeL combined with DRL, a fundamental parameter is the choice of the exploration policy by which the agent interacts with the

environment, which is an absent aspect in classification tasks. For instance, the authors of [36] develop a MeL model where prior information is used to define agnostic exploration policies enabling a better agent adaptation to multiple learning problems.

In the context of CL and MeL for 5G and 6G network management, drift detection is a critical task: if the environment changes abruptly, the MeL paradigm requires the learner to first become aware of the change, and delayed detection may lead to a violation of the service requirements [37]. Drifts can be classified as abrupt or gradual depending on the period during which the system performance degrades. The literature presents several drift detection algorithms, usually considering the prediction error of the learning model for estimating environment changes. It is possible to consider both heuristic [38] or learning-based strategies [39], with different advantages and drawbacks in terms of, e.g., false alarm probability and assumptions on the drift characteristics.

Despite the numerous works investigating CL and MeL in supervised and reinforcement learning scenarios, to the best of our knowledge none of them considers the trade-off between efficiency and effectiveness proposed in this manuscript. In the following sections, we will take advantage of solutions inspired by the literature, analyzing their impact in terms of both training efficiency and user performance. We will consider a baseline CL system where the models trained for previous tasks are stored in a central memory, similarly to what is done in [15]. Besides, we will consider a simple drift detection algorithm to monitor the environment statistics and trigger the retraining. We chose relatively simple techniques for both drift detection and CL in order to focus on the main aspect of our analysis, which is the trade-off between efficiency and effectiveness in resource allocation problems, as represented by the cost of learning.

III. SYSTEM MODEL

Let us consider a generic resource allocation problem, which is modeled as an infinite horizon Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathbf{P}, R, \gamma)$: $\mathcal{S}$ represents the state space, $\mathcal{A}$ is the action space (which is potentially different for each state), $\mathbf{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability matrix, which depends on the current state, the action chosen by the agent, and the landing state, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. Time is divided into slots, and the slot index is denoted by $t \in \mathbb{Z}^+$. The ultimate objective of a DRL agent is to find the optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$, which maximizes the expected long-term reward:

$$\pi^* = \underset{\pi:\mathcal{S}\to\mathcal{A}}{\arg\max}\, \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(\mathbf{s}_t, \pi(\mathbf{s}_t), \mathbf{s}_{t+1})\right]. \tag{1}$$

In our model, at each time slot t, it is possible to allocate N resource units. A resource unit may represent a time-frequency resource in an Orthogonal Frequency Division Multiple Access (OFDMA) communication channel, a specific number of CPU cycles, or a certain amount of power in the device's battery. Particularly, the nature of these resources is immaterial to our model, although it might affect the definition of the MDP. Each resource unit may be used to fulfill a generic *user request*, which may refer to the transmission of a packet or the execution of some computational tasks, depending on the nature of the scenario.

For simplicity's sake, our model assumes that resource units are interchangeable, and each request requires exactly a single resource unit. Future extensions of this work may consider more complex resource allocation schemes, in which requests may require multiple resource units depending on their class and originating user. However, we observe that this can be re-conduced to the basic model by splitting these complex requests into multiple basic requests, returning to the one-to-one mapping.

The system resources are assumed to be partitioned among M different *slices*, where a slice may include a single user, or a group of users with the same features. The action space then contains all possible resource allocation vectors that split the N resources among the M slices:

$$\mathcal{A} = \left\{ \mathbf{a} \in \{0, \dots, N\}^M : \sum_{m=1}^{M} a_m = N \right\}. \tag{2}$$

Furthermore, we assume that each slice is associated with a First-In First-Out (FIFO) queue of requests: each queue has a limited size Q, after which the system starts dropping older requests for that slice to make room for newer ones.

In this work, we assess the system performance in terms of QoS degradation in the different slices. Specifically, the QoS is tied to the latency and reliability with which the user requests are accomplished, according to the slice to which users belong. Assuming that the i-th request from slice m is generated at time $t_{m,i}$, we define the age $\Delta_{m,i}(t)$ of such request as:

$$\Delta_{m,i}(t) = t - t_{m,i}. \tag{3}$$

Hence, the latency of the i-th request from slice m is given by the age $\Delta_{m,i}(t^*)$ at the time t^* the request gets assigned a resource.

Given a specific state configuration $\mathbf{s} \in \mathcal{S}$, we denote by $\Delta_{q_m(i)}$ the latency of the request in position i of the m-th queue. Hence, the system performance is determined by the reward

$$R(\mathbf{s}, \mathbf{a}, \mathbf{s}') = \sum_{m=1}^{M} \sum_{i=1}^{a_m} f_m\left(\Delta_{q_m(i)}\right), \tag{4}$$

where $f_m : \mathbb{N} \to [0, 1]$ is a function mapping the latency of each request to slice m's resulting QoS. With a slight abuse of notation, we set $f_m(\varnothing) = 0, \forall\, m$, where $\varnothing$ indicates that no request is in that position in the queue.

In the case of slices with *hard* QoS requirements, the function $f_m(\cdot)$ returns 1 if the target request is served within the maximum latency permitted by the slice, and 0 otherwise: this may be the case of slices for critical applications,

such as remote surgery or emergency communication services. Conversely, slices with *soft* QoS requirement, may be associated with smoother QoS functions, whose output monotonically decreases as the target request spends more time in the queue: this may refer to slices to serve multimedia or high-throughput applications, such as bulk data transfer.

We can distinguish between *rejected* and *dropped* requests: the first refer to new requests that find a full queue and are immediately discarded, the second refer to queued requests whose waiting time is higher than the deadline. In the latter case, the requests are dropped before service since they would not contribute to the QoS of the slice and just waste resources. We remark that only slices with hard timing requirements can experience dropped requests, while request rejection can occur in any slice.

In our model, user requests being dropped or rejected from the queue have an infinite latency by definition. Besides, it should be noted that dropped or rejected requests do not generate any rewards, as they are never included in the sum. The state of the system is then represented by the age of each request contained in each queue so that in the most general case, $\mathcal{S} = (\{\varnothing\} \cup \mathbb{N})^{M \times Q}$.

The objective of the learning agent is to learn how to allocate resources among slices to maximize the QoS associated with the user requests. It should also be aware of the slices that have a higher risk of violating hard timing requirements and schedule resources to avoid missing deadlines. However, learning is not a cost-free process, and the DRL agent may take up some of the same network resources required by the users in order to improve its policy. As we highlighted in our previous work [11], considering the cost of learning can lead to significantly different choices, limiting the amount and type of experience samples that are selected for training: this is also true regardless of the type of resource the learning requires.

However, that work only considered static policies, which set up a separate virtual channel (either divided in time or in frequency) for the learning data, strictly separating the learning and data planes. Equivalently, an agent learning how to schedule tasks in an edge server could reserve a certain percentage of computation time for self-improvement, but the amount was decided in advance. This is clearly suboptimal: intuitively, the relative returns from policy self-improvement decrease over time, as the agent gradually converges to the optimal policy. After convergence, and as long as the environment statistics are stable, the value of further improvements to the policy is zero by definition. A dynamic policy for adapting the allocation between requests and learning should then take this into account.

Furthermore, the current state of the system also needs to be considered: if delaying the queued requests further does not have a large impact on the QoS, the system can take away resources from the slices to improve the resource allocation policy, but if the impact is big, e.g., if some requests from a slice with hard timing requirements are already close to the deadline, they need to be prioritized, choosing immediate gains over potential future improvements. This is particularly important for non-stationary environments, in which the *coherence time* of the MDP statistics is finite. In such a case, the learning agent needs to adapt to the changing statistics of the environment, and cannot rely on offline training, but must keep learning from experience and adapt to the changes proactively.

IV. LEARNING SOLUTION

In our model, the allocation scheme should address the resource demands of both users and the DRL agent orchestrating the system itself. Theoretically, the decision to allocate resources for the training could be included in the agent's action space. In such a case, the DRL agent needs also to learn when to allocate resources to policy improvement, i.e., identifying the states in which learning is more profitable. However, as the policy evolves over time, learning becomes less of a priority, and the reward associated with the agent's resource demands is reduced. This makes the environment non-stationary and may prevent the agent's policy from converging to the optimal solution. To avoid this issue, it is necessary to separate the problems of orchestrating the data and learning planes.

A. LEARNING AND DRL SLOT PARTITION

We model the environment so that each time slot could be associated with one of two categories, namely, *DRL* and *learning*. In DRL slots, all the resources are allocated to the data plane and are split between the slices according to the action chosen by the DRL agent, while in learning slots, the resources are divided between the learning and data planes following a simple heuristic. The learning slots are not considered as experience samples for the DRL training, as the actions in these slots might not belong to the action space $\mathcal{A}$ considered by the agent if part of the resources are allocated to the learning plane.

Practically, during the learning slots the resource allocation actions $\mathbf{z}$ are in the wider space $\mathcal{Z}$:

$$\mathcal{Z} = \left\{ \mathbf{z} \in \{0, \dots, N\}^M : \sum_{m=1}^{M} z_m \leq N \right\}. \tag{5}$$

We remark that $\mathcal{Z}$ is not the action space, but a super-set of it, i.e., $\mathcal{A} \subseteq \mathcal{Z}$. The definition of $\mathcal{A}$ given in (2) only considers configurations assigning all of the available resource units to the M slices on the data plane, as denoted by the condition $\sum_{m=1}^{M} a_m = N$. Instead, $\mathcal{Z}$ includes actions that allocate part of the resources to improving the agent policy, i.e., to the learning plane: in particular, $N - \sum_{m=1}^{M} z_m$ resource units are used for the transmission of learning data. Naturally, valid actions in $\mathcal{A}$ are also part of $\mathcal{Z}$, and represent the special case in which the learning plane is assigned no resources.

Fig. 1 shows a basic schematic of the learning control loop in a backhaul communication scenario: two classes of users, corresponding to Internet of Things (IoT) (green) and human communications (blue), transmit over a shared link, and the

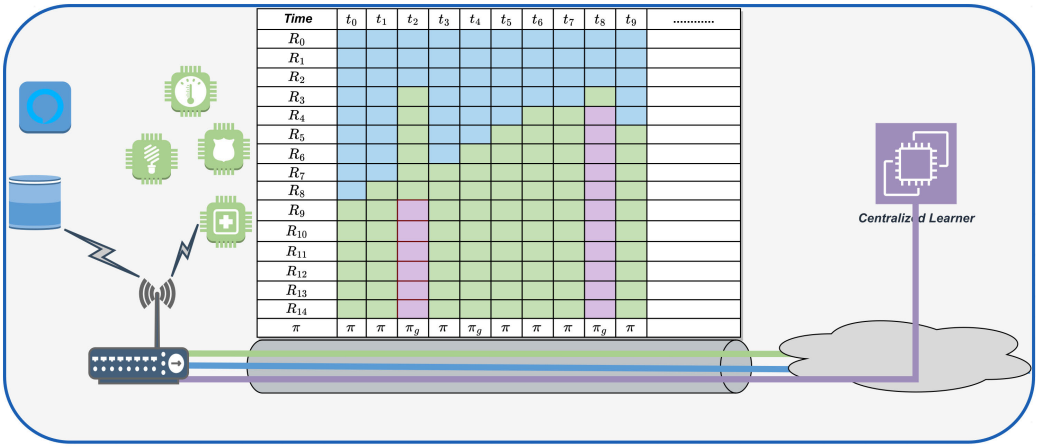


FIGURE 1. Schematic of the learning control loop in a backhaul communication scenario. The connection between the base station and the Internet is used to carry both the users' traffic (blue and green streams) and the data needed to train the learner in the cloud (purple stream), according to the allocation scheme depicted above the link.

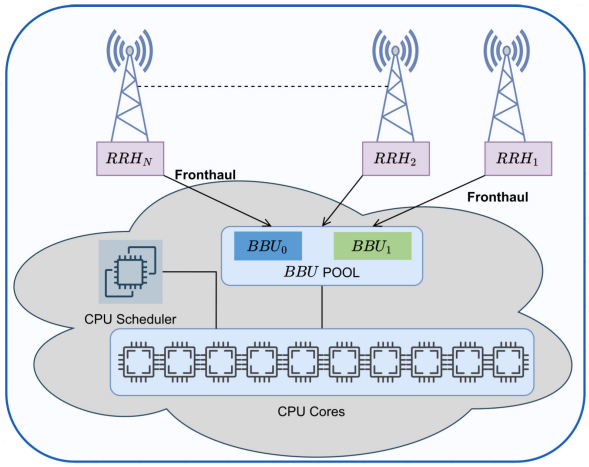


FIGURE 2. Schematic of the computational resource allocation problem in a 5G RAN. The CPU cores can be assigned to different BBUs in order to process packets to and from the corresponding RRHs, or to the learning plane, i.e., to improve the CPU scheduler itself.

resources in each time slot (which correspond to bandwidth and time resources in the uplink to the Cloud) are allocated following a dynamic division. In the case of learning slots, such as slots 3 and 6 of the figure, a significant portion of the resources is allocated to the learning plane (purple line).

We remind the reader that the scenario reported in Fig. 1 is not the only application of our model. For instance, our model may be used to orchestrate the computational resources shared between multiple virtualized Baseband Units (BBUs), each of which is connected to a Remote Radio Head (RRH) through a high-speed link in a 5G Radio Access Network (RAN), as depicted in Fig. 2. In such a case, the RRHs are responsible for transmitting and receiving radio signals, and the virtualized BBUs handle the processing of base-band signals. Processing packets to or from an RRH then becomes a computational task that requires resources, i.e., CPU cycles, to be assigned to the BBU connected to that RRH. In this case, the competition for resources between the data and learning planes is not over spectrum but rather over computing cycles.

For the sake of simplicity, we assume that each slot will be used as a learning slot with probability $\rho(t)$, which decreases linearly over time as the learned policy becomes more stable. The actual function determining $\rho(t)$, i.e., the learning curve shall be defined based on the coherence time of the scenario, i.e., the number of slots τ over which the statistics of the environment will be approximately stationary. As explained later (see (9)) here we choose a linear function, but other choices are possible.

We observe that the value of $\rho(t)$ does not affect the environment explored by the DRL agent, since the training considers only the experience samples from the DRL slots. At the same time, increasing $\rho(t)$ can speed up the process by which the experience samples are exploited for the training. Conversely, lower values for $\rho(t)$ make it more likely that the resource allocation is governed by the agent decisions. In the following, we also propose an adaptive strategy to update $\rho(t)$ based on the training performance of the DRL agent.

B. GREEDY ALLOCATION STRATEGY

In general, we could implement a learning-based strategy for orchestrating the learning slots, e.g., by defining an additional DRL agent that splits the network resources among the user and the learning data plane. However, this would make the system require additional resources for training the new strategy: the initial problem of how to tune the network resources for the transmission of learning data presents itself again, leading to a *recursive* problem. To avoid this issue, we adopt a pre-determined solution that does not need to be tuned according to the learning environment.

Our solution considers two contrasting objectives: minimizing the loss of QoS for users and maximizing the number

of experience samples that can be exploited for the training process. To capture the first aspect, we define a function $\hat{R} : \mathcal{S} \times \mathcal{Z} \rightarrow \mathbb{R}$ that represents an approximation of the instantaneous reward for each resource allocation, considering only the information available in the current state. Assuming that the QoS functions $\{f_m(\cdot)\}$ are known, the value of $\hat{R}(\mathbf{s}, \mathbf{z})$ is:

$$\hat{R}(\mathbf{s}, \mathbf{z}) = \sum_{m=1}^{M} \left[\sum_{i=1}^{z_m} f_m\left(\Delta_{q_m(i)}\right) - \sum_{j=z_m+1}^{Q} f_m\left(\Delta_{q_m(i)} + 1\right) \right]. \tag{6}$$

Note that maximizing (6) may lead to suboptimal resource allocations, as $\hat{R}(\mathbf{s}, \mathbf{z})$ does not account for the long-term reward.

To model the second aspect, we consider that each DRL slot generates an experience sample, which requires ℓ requests (and as many resources) to be transferred to the learner. Due to memory limitations, we assume that the number of samples that can be buffered cannot exceed E. We hence define a second function $S(\mathbf{z}, e)$ that captures the effectiveness of the allocation $\mathbf{z}$ in transferring the $e \in \{0, \ldots, E\}$ samples observed by the agent. Particularly, we define S in terms of the number of experience samples that we manage to transmit in the learning slot but, once again, this is a arbitrary choice and other options might be more suitable for other scenarios. The greedy strategy is then the solution to the following optimization problem:

$$\mathbf{z}^*(\mathbf{s}, \mathbf{z}, e) = \arg\max_{\mathbf{z}} \left(M^{-1} \hat{R}(\mathbf{s}, \mathbf{z}) + E^{-1} S(\mathbf{z}, e) \right). \tag{7}$$

If f_m is concave for all slices with a soft timing requirement, we can exploit the FIFO nature of the queue to provide a simple iterative solution. Practically, we start from the empty assignment and gradually allocate resources to either one of the slices or the learning process, depending on the value of the utility function.

Fig. 3 shows a schematic of the full resource allocation strategy during a learning slot, in a simple case with $M = 2$. The state s_t of the system, observed by the learning agent, is given by the number of queued requests associated with each slice. At each time step, the node randomly chooses whether to exploit the policy π of the learning agent or (with probability $\rho(t)$) the greedy allocation policy, denoted by π_g. The latter policy reserves some resources for the learning plane and, therefore, receives as input the number of experience samples that have been observed by the agent. The output of both the agent and greedy strategy is the resource allocation vector $\mathbf{z} = [z_L, z_1, \ldots, z_M]$, where, in case π is used, we always have $z_L = 0$. Finally, we observed that, whenever the agent policy π is followed, a new experience sample $(s_{t-1}, a_{t-1}, r_t, s_t)$ is generated. Hence, selecting π increases the number of learning requests, while π_g makes it possible to satisfy such requests, improving the agent policy π. Only by accurately tuning the probability $\rho(t)$ it is possible to maximize the training efficiency.

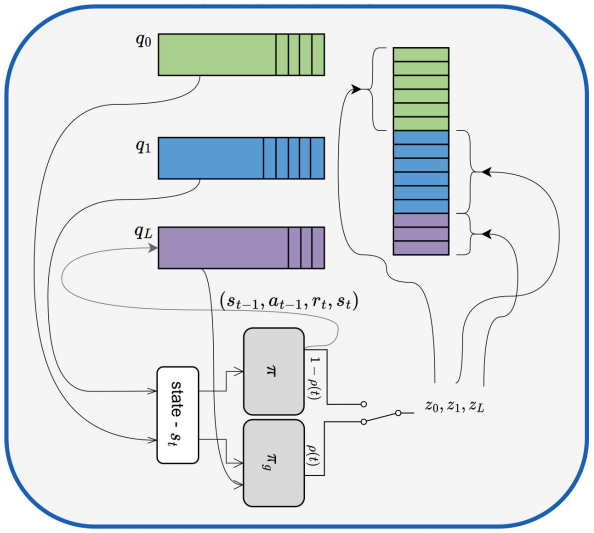


FIGURE 3. System-level diagram of the proposed scheme.

C. CONTINUAL LEARNING

Differently from our previous work [11], our solution can also adapt to drift in the environment statistics. This allows us to handle scenarios where the data traffic in each slice changes in time because, e.g., the number of users associated with each slice changes as well. Practically, we assume the environment can be characterized by a set of parameters $\boldsymbol{\omega}$, which determine its stochastic dynamic. From time to time, this parameter can instantaneously change, making the environment non-stationary (in the DRL jargon, changing the task) and requiring the strategy to be updated in order to pursue the new task.

To cope with such a non-stationary context we proposed a CL strategy similar to the work in [15], but including considerations on the cost of learning. When a context-change is detected, say from $\boldsymbol{\omega}$ to $\boldsymbol{\omega}'$, we consider its significance by means of a distance function $\eta(\boldsymbol{\omega}, \boldsymbol{\omega}')$: if it is larger than a threshold η_{thr}, we consider the environment to be novel enough to warrant retraining. If the change is smaller than the threshold, we implicitly assume that the policy is close enough to the optimum that maintaining it is more convenient than running a new training phase.

We define the environment index $k \in \mathbb{N}$, which starts from 0 and is incremented at every significant change in the environment. Our solution maintains a record of the past environments and the respective learned policies, so that as the environment shifts into the new context $\boldsymbol{\omega}_{k+1} = \boldsymbol{\omega}'$, we can find the closest past environment:

$$j^* = \underset{j \in \{0,\ldots,k\}}{\text{argmin}}\ \eta(\boldsymbol{\omega}_j, \boldsymbol{\omega}'). \tag{8}$$

If the previously experienced environment is close enough to the new one, i.e., $\eta(\boldsymbol{\omega}_{j^*}, \boldsymbol{\omega}') < \eta_{\text{thr}}$, we can apply the stored policy directly, relying on a short training phase with increased exploration rate and training probability $\rho(t)$ to adapt to the small change. Instead, if no environment in

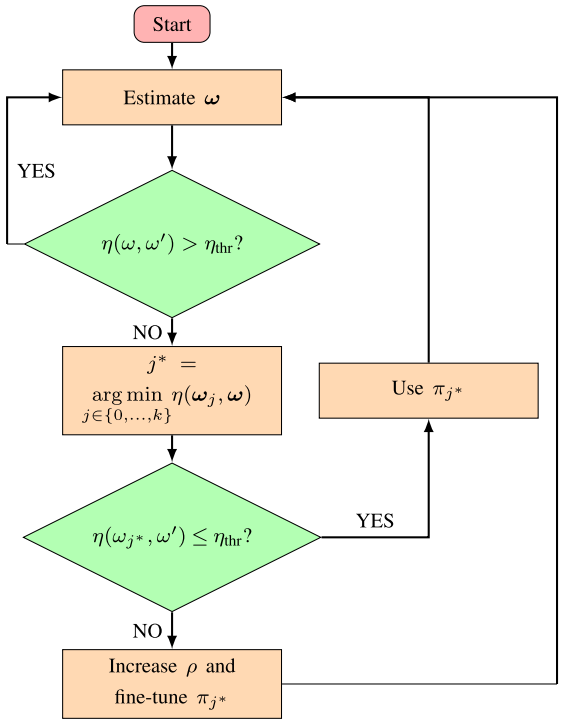


FIGURE 4. Flowchart of the context adaptation algorithm.

the memory is close enough, training needs to begin from scratch, leading to a slower and more expensive training phase. A flowchart of the above process is depicted in Fig. 4.

After a drift detection, we empirically set the training probability $\rho(t)$ to an initial value ρ_0. Then, we smoothly decrease the value of $\rho(t)$ up to minimum value ρ_f, making the system reduce the number of network resources allocated for the agent training, as the agent policy is improved. We consider two possible configurations for dynamically tuning $\rho(t)$, named *constant* and *adaptive decay*, respectively. In the *constant decay* scenario, $\rho(t)$ decreases exponentially, and $\rho(t)$ is updated every H_0 slots as follows:

$$\rho(t) = \max\left(\rho_f, \rho(t-1) \cdot \sigma_0\right). \tag{9}$$

In the adaptive scenario, the probability $\rho(t)$ is adjusted based on the Temporal Difference Error (TDE), i.e., the difference between the q-values computed by the agent policy and the q-values updated according to the adopted Reinforcement Learning (RL) algorithm. Assuming that the agent is trained according to the classical Q-learning framework, the TDE is defined as:

$$\lambda(s, a, r, s') = \left| Q(s, a) - \left(r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') \right) \right|, \tag{10}$$

where $Q(s, a)$ is the estimated long-term reward of the learning agent when taking action a in state s. To stabilize the adjustment of $\rho(t)$ and avoid frequent fluctuations, we compute the moving average $\bar{\lambda}$ of the TDE considering a period of D slots. Hence, $\rho(t)$ is exponentially decreased if the moving average of the TDE is decreasing and exponentially increased if the moving average of the TDE is increasing or remains constant:

$$\rho(t) = \begin{cases} \max\left(\rho_f, \rho(t-1) \cdot \sigma_1\right), & \text{if } \bar{\lambda}(t) < \bar{\lambda}(t-D); \\ \min\left(\rho_0, \rho(t-1) \cdot \dfrac{1}{\sigma_1}\right), & \text{if } \bar{\lambda}(t) \geq \bar{\lambda}(t-D). \end{cases} \tag{11}$$

Doing so, the adaptive decay method aims at regulating the value of $\rho(t)$ every H_1 slots according to the estimation of the accuracy of the agent policy.

D. COMPUTATIONAL AND MEMORY COMPLEXITY

The computational complexity of the feedforward Deep Q-Network (DQN) [40] is $\mathcal{O}(W)$, where W is the total number of weights in the neural network approximating the long-term reward function. This value is assumed to be relatively small, as the complexity of resource allocation tasks does not usually require millions of parameters, like some tasks based on images or other high-dimensional data. On the other hand, running back-propagation on m experience samples requires $\mathcal{O}(mW)$ operations, which may be beyond the computational capabilities of Edge devices.

The proposed heuristic to decide whether to use resources to train the DQN takes $\mathcal{O}(MQ)$ operations, and $MQ \ll W$ for most practical systems. Greedy allocation steps are then extremely computationally lightweight, requiring fewer operations than the normal operation of the DQN. The proposed heuristic does not incur in the same pitfall as learning-based approaches, as it follows a relatively simple rule and does not add to the computational burden of the Edge node.

On the other hand, the memory requirements for the CL solutions might be more significant: if we maintain up to K environmental models as a starting point, we will incur a memory cost of $4KW$ bytes to save all DQN parameters as single-precision floating point numbers. This memory cost might be significant, but it can be tuned: as we will show in Sec. VI, keeping track of more past environments can result in better performance, but also gives diminishing returns. Therefore, it is possible to tune the value of K and η_{thr} to limit the overall memory requirements.

V. NETWORK SLICING USE CASE

To substantiate the approach on a practical but easy-to-analyze use case, we consider the resource allocation problem in a simple network slicing scenario. We assume a common communication link is used to transmit both the data packets generated by the users, which belong to two different network slices, and the pieces of information used to feed the learner. Time is divided in slots of constant duration τ, and in each slot the transmission channel can carry N orthogonal and identical resource units. The scenario fits the general model presented in the previous section, as the communication resources are shared between the data and learning planes. The full parameters for the scenario, which we will describe in this section, are given in Table 1.

TABLE 1. Use case and learning parameters.

Parameter	Symbol	Value
Communication system		
Number of subchannels	N	15
Slot time duration	τ	1 ms
Packet queue length	Q	1500
Packet size	L	512 B
Link capacity	C	7.68 MB/s
Learning plane		
Discount factor	γ	0.95
Learning queue length	E	1500
Packets required for each sample	ℓ	3
Initial learning slot probability	ρ_0	0.2
Final learning slot probability	ρ_f	0.01
Static Learning slot decay pace	H_0	1 s
Adaptive Learning slot decay pace	H_1	100 ms
Queue pressure parameter	χ_1	1400
Static ρ decay parameter	σ_0	0.98
Adaptive ρ decay parameter	σ_1	0.95
Moving average parameter	D	100 ms
Environment Drift Threshold	η_{thr}	0.5

A. COMMUNICATION SYSTEM MODEL

We consider two slices, corresponding to the two types of data sources:

- Slice 1 is for bulk file transfer, for which we do not set any strict latency constraints. However, we want the system to have the highest possible reliability to ease the burden on the higher layers. As such, $f_1(T) = 1$ for all finite values of T, but the QoS is 0 if T is infinite (i.e., if the packet is dropped);
- Slice 2 is intended for interactive traffic, such as video conferencing or Virtual Reality (VR) traffic, with a strict latency deadline: packets need to be transmitted with a maximum latency $T^{(2)}_{\text{soft}}$. For the sake of simplicity, we assume that, after $T^{(2)}_{\text{soft}}$, the utility decreases linearly with time, dropping to 0 if the latency is higher than $T^{(2)}_{\text{max}} \geq T^{(2)}_{\text{soft}}$, i.e., $f_2(x) = 1$ if $0 \leq x \leq T^{(2)}_{\text{soft}}$, and $f_2(x) = \max\left(0, 1 - (x - T^{(2)}_{\text{soft}})/(T^{(2)}_{\text{max}} - T^{(2)}_{\text{soft}})\right)$ if $x > T^{(2)}_{\text{soft}}$.

We remark that, although these QoS functions are reasonable, they may not be the most appropriate to represent the considered slices. Since the purpose of this study is to gain insights on the cost of learning in dynamic systems, more than proposing a quantitative performance analysis of the use-case, we prefer these neatly-shaped functions that allow for a qualitative performance analysis while easing the interpretability of the results.

The number of active users in each slice is variable, making traffic non-deterministic. We consider a maximum number of active users $U_m \in \mathbb{N}$ for each slice $m \in \{1, 2\}$. Each user follows a on-off model, which can be modeled as a Gilbert-Elliott binary Markov chain with transition probability matrix $\mathbf{O}^{(m)}$. In state 0, the user does not transmit, while in state 1, it transmits packets of size L with a constant bitrate R_m.

The aggregate traffic generated by slice m is then represented by the number $u_m(t)$ of active users at time t, multiplied by R_m. We can then define a Markov chain over $u_m \in \{0, \ldots, U_m\}$, with transition probabilities:

$$P(u_m(t+1) = v | u_m(t) = u) = \sum_{w=\max(0,u+v-U_m)}^{\min(u,v)} (O^{(m)}_{11})^w (O^{(m)}_{10})^{u-w} \times \binom{u}{w} \binom{U_m - u}{v - w} (O^{(m)}_{01})^{v-w} (O^{(m)}_{00})^{U_m-u-v+w}. \tag{12}$$

The expected traffic G_m from slice m can be computed as:

$$\mathbb{E}[G_m] = \frac{O^{(m)}_{01} U_m R_m}{O^{(m)}_{01} + O^{(m)}_{10}}. \tag{13}$$

On the other hand, the total channel capacity is simply:

$$C = \tau^{-1} NL. \tag{14}$$

With the values in Table 1, we obtain $C = 7.68$ MB/s. Some representative environment parameters are reported in Table 2. Note that, based on the definition, slice 1 can only experience rejected packets, while slice 2 can have both rejected and dropped packets (if their age exceeds the deadline $T^{(2)}_{\text{max}}$).

B. LEARNING PLANE

In this part we define the two components of the learning plane, i.e., the DRL agent, which will assign resources during the DRL slots, and the greedy split approach, which manages resource allocation in the learning slots.

DRL AGENT SETTINGS

We use a DQN [40] for the agent, as the problem is simple enough not to require more advanced architectures.

We consider a simplified state: for each slice $m \in \{1, 2\}$, the input to the network is given by the following values:

- The number $q_m \in \{0, \ldots, Q\}$ of packets in the queue;
- The minimum latency T^{min}_m for packets transmitted in the previous slot;
- The maximum latency T^{max}_m for packets transmitted in the previous slot;
- The average latency T^{avg}_m for packets transmitted in the previous slot;
- The number d_m of discarded (dropped or rejected) packets in the previous slot;

TABLE 2. Traffic model parameters.

Env. Index	Policies		Slice 1 Parameters				Slice 2 Parameters						
	ω_{start}	ω_{end}	U_1	R_1	$\mathbf{O}^{(1)}$	$\mathbb{E}[U_1]$	U_2	R_2	$\mathbf{O}^{(2)}$	$\mathbb{E}[U_2]$	$T^{(2)}_{\text{soft}}$	$T^{(2)}_{\max}$
$\boldsymbol{\omega}_0$	$\theta \sim \mathcal{N}(0, 0.1)$	θ_{ω_0}	28	512kB/s	$\begin{pmatrix} 0.617 & 0.382 \\ 0.544 & 0.455 \end{pmatrix}$	11.561	5	512kB/s	$\begin{pmatrix} 0.156 & 0.843 \\ 0.763 & 0.236 \end{pmatrix}$	2.625	50ms	70ms
$\boldsymbol{\omega}_{12}$	θ_{ω_1}	$\theta_{\omega_{12}}$	15	512kB/s	$\begin{pmatrix} 0.158 & 0.841 \\ 0.218 & 0.781 \end{pmatrix}$	11.908	30	512kB/s	$\begin{pmatrix} 0.925 & 0.074 \\ 0.672 & 0.327 \end{pmatrix}$	2.976	50ms	70ms
$\boldsymbol{\omega}_{102}$	$\theta_{\omega_{23}}$	$\theta_{\omega_{102}}$	20	512kB/s	$\begin{pmatrix} 0.797 & 0.202 \\ 0.316 & 0.683 \end{pmatrix}$	7.810	83	512kB/s	$\begin{pmatrix} 0.949 & 0.050 \\ 0.547 & 0.452 \end{pmatrix}$	6.944	50ms	70ms
$\boldsymbol{\omega}_{110}$	$\theta_{\omega_{75}}$	$\theta_{\omega_{110}}$	9	512kB/s	$\begin{pmatrix} 0.696 & 0.303 \\ 0.156 & 0.843 \end{pmatrix}$	5.937	37	512kB/s	$\begin{pmatrix} 0.804 & 0.195 \\ 0.620 & 0.379 \end{pmatrix}$	8.870	50ms	70ms

- The current number a_m of resource units allocated to the slice.

The values for each queue are contained in the tuple $\mathbf{s}^{(m)} = (q_m, T_m^{\min}, T_m^{\max}, T_m^{\text{avg}}, d_m, a_m)$, to which we add another parameter $\xi^{(m)}$, i.e., the difference in the utility for slice m if packets are not transmitted in the next slot. For slice 1, i.e., the latency-insensitive one, this corresponds to the expected number of rejected packets; for the second slice, it is applicable if some packets are close to or over the soft deadline $T^{(2)}_{\text{soft}}$. If the head-of-line packets are close to $T^{(2)}_{\max}$, this can even lead to packet drops. We can define it as follows:

$$\xi^{(2)} = \sum_{i=1}^{q_m} f_2\left(\Delta_{q_2(i)}\right) - f_2\left(\Delta_{q_2(i)} + 1\right). \tag{15}$$

As the first slice does not have latency requirements, there are no equivalent parameters for it. All the input values are normalized to fit in the range between 0 and 1.

The input to the DQN is then given by $\mathbf{s}^{(m)}, \xi^{(m)}$ which corresponds to a total of 13 values; the training parameters are defined in Table 1.

We denote the resource allocation vector during slot t by $\mathbf{a}_t = [a_1, a_2]$. At each step, the DRL makes an action δ_t to change the resource allocation as:

$$\mathbf{a}_{t+1} = \mathbf{a}_t + \delta_t. \tag{16}$$

For the sake of simplicity and interpretability of the results, we admit only actions $\delta_t \in \{(1, -1), (0, 0), (-1, 1)\}$ that change the allocation to each slice of at most 1 resource unit per step. The outputs of the DQN correspond to the estimated long-term value of selecting each δ_t, so the network only has 3 output values. The network architecture is given in Table 3.[1]

Greedy split algorithm settings

We set the size of the experience sample queue $E = 1500$, and implement an early rejection policy. When a sample is generated, its rejection probability is equal to $\frac{e}{E}$, i.e., to the current pressure on the queue. Consequently, samples that find a full queue are always rejected, but sometimes samples that could fit in the queue are dropped in favor of new experiences, avoiding too many correlated samples filling the queue.

[1]The complete implementation of the DQN agent and dynamic resource allocation is available at https://github.com/slahmer97/costoflearning

TABLE 3. DQN architecture.

Layer inputs	Layer Outputs	Activation function
13	64	ReLU
64	32	ReLU
32	3	Linear

For allocating network resources within the user slices and the learning data plane, the greedy strategy π_g operates as follows. As the first slice has no latency requirements, we consider allocating resources to it greedily only when the number of packets in the queue is higher than a threshold χ_1: in this way, we avoid packet rejections but also leave more resources for learning plane and latency-sensitive packets.

We can then define the following estimated rewards:

$$\hat{R}_1(\mathbf{s}, \mathbf{z}) = \min(0, z_1 - \min(q_1 - \chi_1, N)); \tag{17}$$

$$\hat{R}_2(\mathbf{s}, \mathbf{z}) = \min(0, z_2 - \min(\xi_2, N)); \tag{18}$$

$$S(\mathbf{z}, e) = -\left(\min(e, N) - \left(N - \sum_{m=1}^{2} z_m\right)\right). \tag{19}$$

The minimum operation ensures that resources will not be allocated to a slice once the queue pressure is below the limit ξ_1 or all packets with a close deadline are served, respectively. We can define the following problem:

$$\mathbf{z}^*(\mathbf{s}, \mathbf{z}, e) = \arg\max_{\mathbf{z} \in \mathcal{Z}} S(\mathbf{z}, e) + \sum_{m=1}^{2} \hat{R}_m(\mathbf{s}, \mathbf{z}). \tag{20}$$

As the problem can easily be converted to an integer linear problem, we can easily solve it through iterative methods.

C. CONTINUAL LEARNING

In our environment, the statistics of the traffic change periodically every 500 seconds: the parameter vector $\boldsymbol{\omega}$ includes $\mathbf{O}^{(1)}$, $\mathbf{O}^{(2)}$, U_1, or U_2, and as a result, corresponding changes are made to the transition matrix $\mathbf{P}$. The policy for a given set of environmental parameters $\boldsymbol{\omega}$ is defined by the set of corresponding trained weights vectors in the DRL neural network, $\theta_{\boldsymbol{\omega}}$.

In the slicing task, the average traffic for each slice is enough to characterize a new environment, and we can identify each context with the vector $\boldsymbol{\omega} = (\mathbb{E}[U_1], \mathbb{E}[U_2])$. Additionally, we define the threshold η_{thr} as the point at which we trigger this event. In other words, we only initiate a change event if the distance between two environments is greater than η_{thr}. While the average may not accurately represent the environment due to potential variance changes with a constant average, in our specific scenario, the average proved sufficient in maintaining a system performance near the ideal one (as described in the following section).

We observe that the implementation of a robust method for detecting changes in the environment is critical, as the failure to identify a true change or the detection of a false change can lead to a degradation in system performance. At the same time, the design of new detection algorithms for such a goal goes beyond the scope of this study, which focuses on analyzing the trade-off between efficiency and effectiveness in CL systems. Hence, we defer the study of reliable context-recognition problems as well as the tuning of hyper-parameters of our framework to future research.

Following the strategy we outlined in Sec. IV-IV-C, the weights of the neural network are chosen from the closest environment observed in the past. We can also make an additional consideration: if the offered traffic is decreasing for both slices, the previous policy will still obtain good results, as the new environment is substantially easier than the previous one. We then define a strict minority relation between vectors, so that $\mathbf{x} \prec \mathbf{y}$ if the two vectors $\mathbf{x}$ and $\mathbf{y}$ are the same length and each element of $\mathbf{x}$ is smaller than the corresponding element of $\mathbf{y}$:

$$\mathbf{x} \prec \mathbf{y} \Leftrightarrow |\mathbf{x}| = |\mathbf{y}| \wedge x_i < y_i, \ \forall i \in \{1, \dots, |\mathbf{x}|\}. \tag{21}$$

We also employ the Euclidean distance to define $\eta(\boldsymbol{\omega}, \boldsymbol{\omega}')$ between two environments, so the centralized agent is updated as follows:

$$\theta_{\boldsymbol{\omega}_{k+1}} = \begin{cases} \theta_{\boldsymbol{\omega}_k}, & \text{if } \boldsymbol{\omega}_{k+1} \prec \boldsymbol{\omega}_k; \\ \theta_{\boldsymbol{\omega}_{j^*}}, & \text{if } \|\boldsymbol{\omega}_{k+1} - \boldsymbol{\omega}_{j^*}\|_2 < \eta_{\text{thr}}; \\ \theta_0, & \text{otherwise}, \end{cases} \tag{22}$$

where $||\mathbf{x}||_2$ is the ℓ_2 norm of vector $\mathbf{x}$ and $\theta_0 \sim \mathcal{N}(0, 0.1)$ is a random initialization vector. After the new weight vector is selected, the algorithm temporarily increases both the training slot probability $\rho(t)$ and the exploration rate of the DRL agent, so the policy can be adapted to the new task.

Algorithm 1 Environment Parameter Update

Output: $\mathbf{O}^{(1)}$, $\mathbf{O}^{(2)}$, $\mathbf{U_1}$, $\mathbf{U_2}$

while $\frac{\mathbb{E}[G_m]}{C} \notin [0.75, 1.1]$ **do**
  for i $\in [1, 2]$ **do**
    $\mathbf{O}^{(i)}[1], [1] = \mathcal{U}(0.05, 0.95)$
    $\mathbf{O}^{(i)}[1], [0] = 1.0 - \mathbf{O}^{(i)}[1], [1]$
    $\mathbf{O}^{(i)}[0], [0] = \mathcal{U}(0.05, 0.95)$
    $\mathbf{O}^{(i)}[0], [1] = 1.0 - \mathbf{O}^{(i)}[0], [0]$
  end for
  $\mathbf{on_0} = \frac{\mathbf{O}^{(0)}[0],[1]}{(\mathbf{O}^{(0)}[0],[1]+\mathbf{O}^{(0)}[1],[0])}$
  $\mathbf{on_1} = \frac{\mathbf{O}^{(1)}[0],[1]}{(\mathbf{O}^{(1)}[0],[1]+\mathbf{O}^{(1)}[1],[0])}$
  $\mathbf{U_1} = \texttt{random}\left(2, \left\lfloor \frac{14}{\mathbf{on_0}} \right\rfloor, 1\right)$
  $\mathbf{U_2} = \left\lfloor \frac{\max(\lfloor 15 - \mathbf{U_1 on_0} \rfloor, 1)}{\mathbf{on_1}} \right\rfloor$
  Compute the resulting $\mathbb{E}[G_m]$
end while
return $\mathbf{O}^{(1)}$, $\mathbf{O}^{(2)}$, $\mathbf{U_1}$, $\mathbf{U_2}$

VI. SIMULATION SETTINGS AND RESULTS

In this section, we present numerical findings that demonstrate the efficacy of the proposed learning framework in a non-stationary environment. Particularly, we first present the settings of the systems as well as the different configurations of the learning framework that we tested. Then, we analyze the results, focusing on the system performance within the different approaches, given in terms of normalized reward and other communication metrics.

A. SETTINGS

We consider the resource allocation scenario presented in Sec. V, and run the resource allocation for 64000 seconds, corresponding to 128 coherence periods lasting 500 s each. Each allocation step corresponds to 1 ms, which makes the environment stable for 5×10^5 steps, then abruptly transiting to a different behavior. The changes in the environment follow Algorithm 1. In the algorithm, we denote the probability of a user belonging to slice m being active as on_m, and the uniform distribution between a and b as $\mathcal{U}(a, b)$. We consider three different CL approaches:

- *From scratch*: the CL approach is not applied, and whenever an environment drift is detected the learning agent's architecture is initialized from scratch, leading to a new training phase. We note that this system is similar to the one in the conference version of this work [13].
- *Previous model*: the learning agent is not retrained from scratch but its starting parameters are the ones that were obtained at the end of the previous coherence period.
- *Nearest model*: the agent neural network parameters are the ones obtained after convergence in the most similar environment among those analyzed so far, according to the CL approach presented in Sec. V-V-C.

In the first two cases (i.e., the from scratch and previous model initialization), we consider the constant decay algorithm for tuning the value of $\rho(t)$ in time, considering

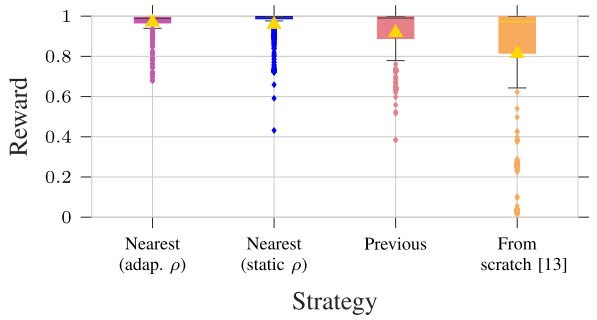


FIGURE 5. Boxplot of the normalized reward with different initialization strategies.

$H_0 = 10$ seconds as updating period. Instead, when using the nearest model for updating the agent policy, we consider both the constant and the dynamic decay algorithm, with the updating period set to $H_0 = 1$ and $H_1 = 0.1$ second, respectively. Independently from the updating methodology for varying $\rho(t)$, we set the maximum and minimum value of $\rho(t)$ to $\rho_0 = 0.2$ and $\rho_f = 0.01$, respectively. Finally, for computing the moving average of the TDE, we consider a period of $D = 1$ second.

For evaluating the learning strategy, we consider three different benchmarks:

- *Out-of-band*: this scheme represents the ideal case in which training data is transmitted over a side channel with infinite capacity. This aligns with the common assumption in the literature of free training, and represents an upper bound for performance;
- *Frequency Division Multiple Access (FDMA)*: here we assume 1 resource unit in each slot is reserved for the learning plane, while the other 14 resource units can be freely allocated to the users' slices;
- *Time Division Multiple Access (TDMA)*: we consider a time division between the learning and data planes, in which all available resources are allocated to the learning plane once every T_ℓ slots. We consider two cases, with $T_\ell = 10$ and $T_\ell = 100$.

The full simulation parameters are listed in Tables 1 and 2.

B. RESULTS

At first, we explore the impact of three different initialization strategies on the normalized reward. Particularly, the normalized reward is equal to 1 if all packets are delivered with utility 1 (i.e., before $T^{(2)}_{\text{soft}}$ if they belong to slice 2) and 0 if all packets are dropped or rejected. Looking at Fig. 5, we can appreciate how the strategy of starting the learning process from scratch, assuming that the new environment is completely new, is significantly outperformed by the other approaches. Exploiting transfer learning from the information collected from past environments is then a critical factor, leading to a higher enhancement in system performance than our earlier work [13]. The CL approach presented in Sec.IV-IV-C, which uses the nearest recorded environment, enables a significant boost over the strategy of keeping the previous environment. This is because the nearest environment selection makes it possible to start the DQN with weights that are already close to the correct ones, speeding up the training phase. Appreciably, using the adaptive algorithm for tuning $\rho(t)$, it is possible to further improve the performance and reduce the number of outliers in the boxplot.

In Fig. 6, we report the total number of active flows (i.e., of users transmitting data in the slot) of the two slices during a simulation time of 2500 s, corresponding to 5 environment drifts. The same simulation period was analyzed in depth in Fig. 7, where we report the normalized size $|Q_i|$, with $i \in \{0, 1\}$, of the queues associated with the two slices, the expectation $\mathbb{E}[R]$ of the normalized reward, the moving average $\overline{\lambda}(t)$ of the TDE, and the training probability $\rho(t)$. In particular, the figure enables the comparison between the results obtained with the adaptive and static methods for tuning the training probability $\rho(t)$ across the five environments.

Fig. 7a-b compare the size of the queues associated with the two traffic classes. The sizes of both the queues are higher and show more variability at the beginning of each coherence period, since the agent's policy still needs to be tuned, stabilizing after approximately 200 seconds. On average, slice s_0 is characterized by a higher number of queued packets, which is consistent with the QoS requirements of the related data sources. Conversely, slice s_1 is associated with a much lower queuing size, reflecting the stronger delay constraint for the second class of traffic. During each coherence period, the expected reward $\mathbb{E}[R]$ follows an opposite trend than the queuing size since it increases as the agent refines its actions up to a maximum value of 1 (Fig. 7c). In most cases, the adaptive algorithm seems to lead to a faster convergence time than the static approach: in all environments but the second, the dynamic adaptation of ρ ensures that the reward is maximized earlier. In the second environment, such a phenomenon does not occur and, particularly, the two approaches achieve two different resource allocation policies after convergence, which is confirmed by the differences between the values of $|\mathcal{Q}_0|$ in the two cases. The different behaviors of the two methods for tuning $\rho(t)$ can be explained by considering Fig. 7d-e. The results outline how the static approach decreases $\rho(t)$ exponentially over time, ensuring that the percentage of resources used for the training gradually reaches the minimum value as the agent improves its knowledge about the system. Instead, the dynamic approach adapts $\rho(t)$ to the variations of $\overline{\lambda}(t)$, leading to an irregular trend where the percentage of training resources may both increase and decrease in time. This allows $\rho(t)$ to reach its final minimum value faster in most environments, but leads to a slower and less stable training. Indeed, in that environment the expected TDE obtained after the agent policy convergences is significantly higher with the adaptive approach, although the performance at convergence is still almost perfect.

For comparing our leaning framework with the benchmark approaches, we sampled four different coherence periods, whose parameters are reported in Table 2. The reported index corresponds to the time of their appearance in the simulation.

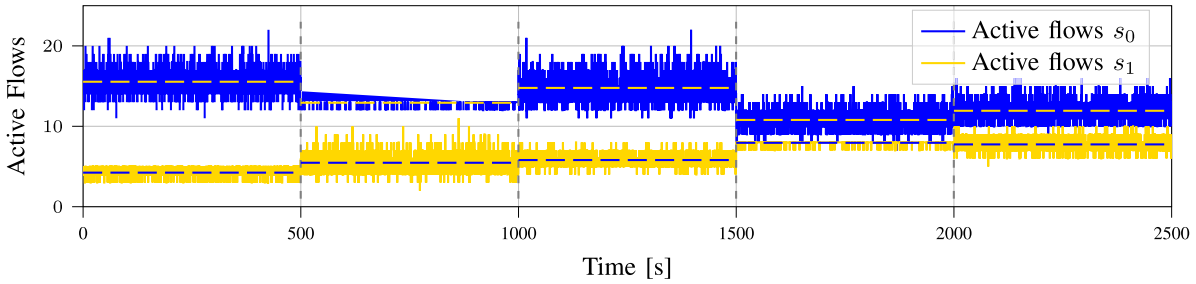
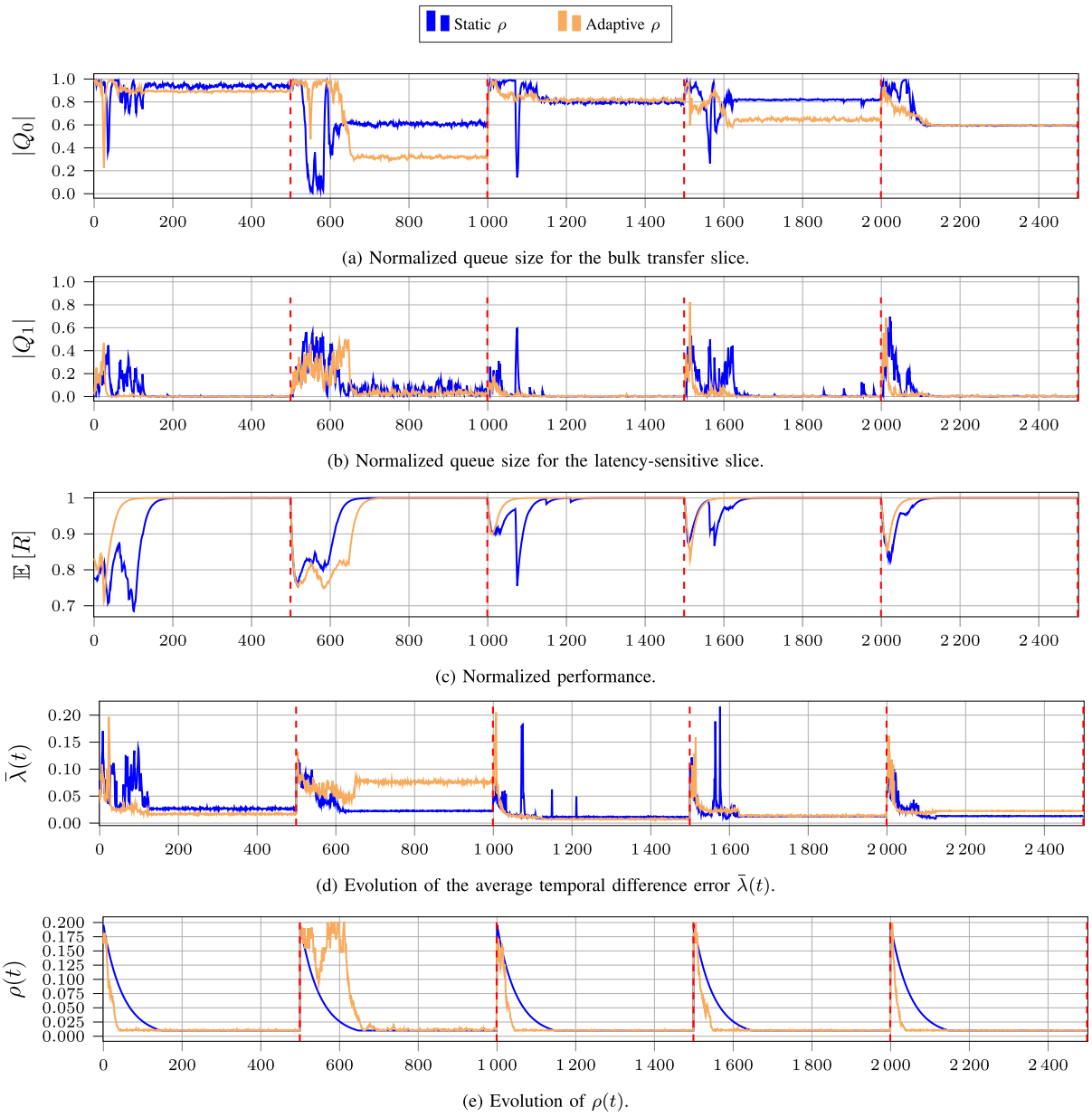


FIGURE 6. **Number of active flows over time in slice s_0 (upper curve) and s_1 (lower curve); the environment statistics change every 500 s.**



(a) Normalized queue size for the bulk transfer slice.

(b) Normalized queue size for the latency-sensitive slice.

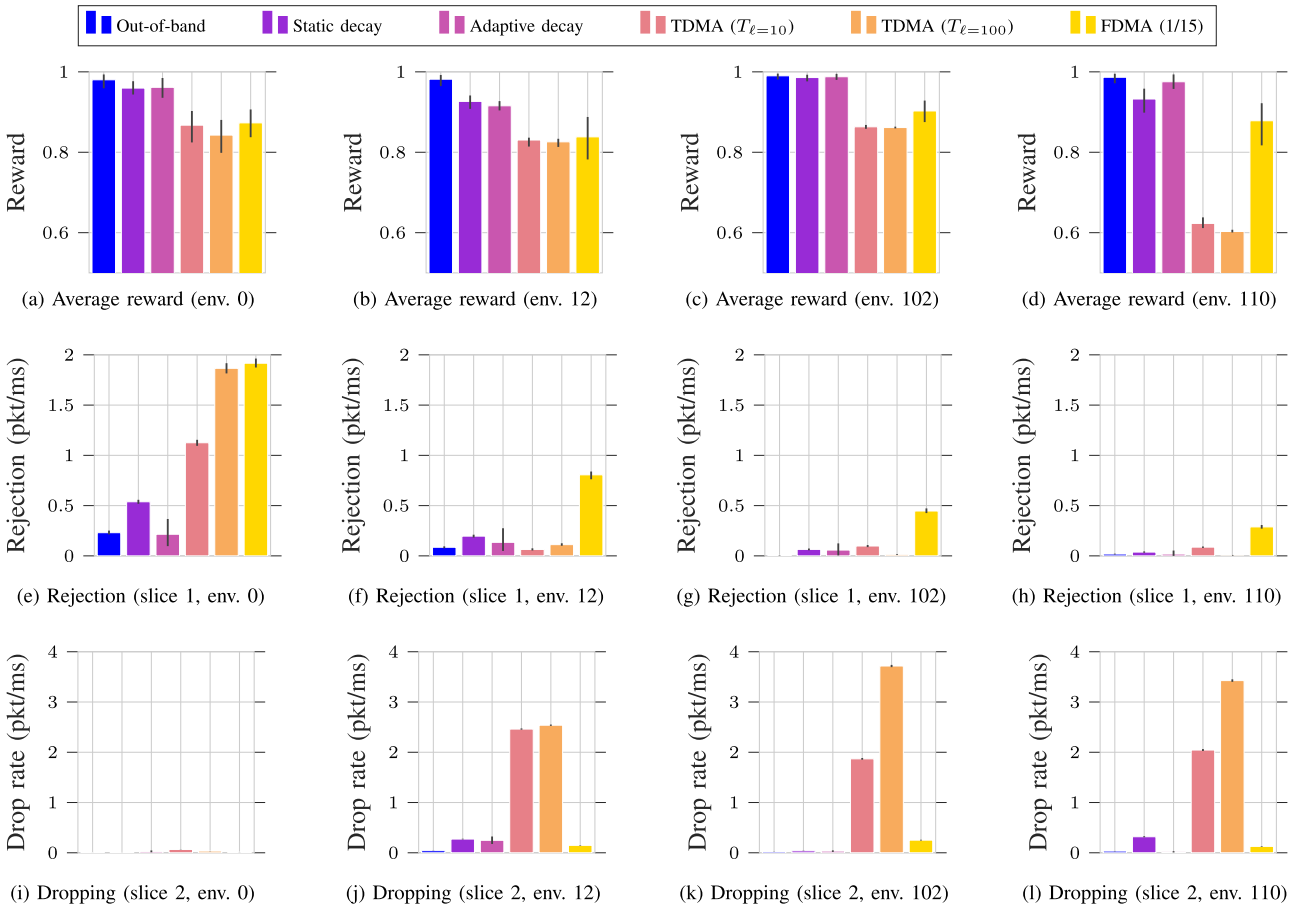
(c) Normalized performance.

(d) Evolution of the average temporal difference error $\bar{\lambda}(t)$.

(e) Evolution of $\rho(t)$.

FIGURE 7. **Comparison between the adaptive and static approaches in the first 2500 s of the simulation.**

In all the selected environments, the load is greater than 0.945, i.e., the offered traffic is very close to the channel capacity, and in env. 12 and 110, the load is around 0.99. Fig. 8 shows the average reward for these environments, along with the



(a) Average reward (env. 0) (b) Average reward (env. 12) (c) Average reward (env. 102) (d) Average reward (env. 110)

(e) Rejection (slice 1, env. 0) (f) Rejection (slice 1, env. 12) (g) Rejection (slice 1, env. 102) (h) Rejection (slice 1, env. 110)

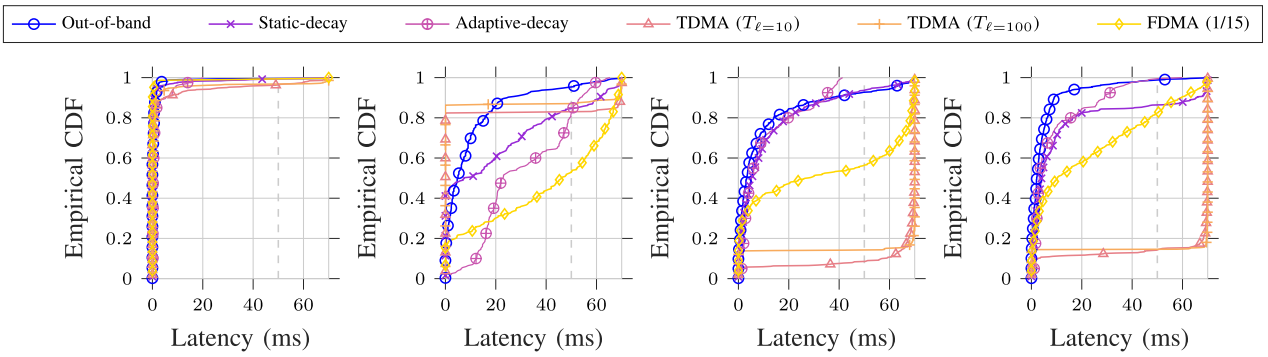
(i) Dropping (slice 2, env. 0) (j) Dropping (slice 2, env. 12) (k) Dropping (slice 2, env. 102) (l) Dropping (slice 2, env. 110)

FIGURE 8. Performance of the schemes in four different sampled environments, measured by the average normalized reward, the packet reject ion rate for slice 1, and the packet drop rate for slice 2.

packet drop rate for slice 1 (i.e., packets exceeding $T^{(2)}_{\max}$, which are dropped from the queue as their utility is 0) and the packet rejection rate for slice 2 (i.e., packets which find a full buffer and are discarded directly). The indices of the periods represent an incremental number of different environments seen by the CL agent: the first, with index 0, is the first to be seen, while there are 12 other periods between 0 and 12, and so on. Each period has the same duration, i.e., 500 seconds.

As a first observation, we note that the ideal out-of-band policy, which neglects the cost of learning, clearly outperforms those that reserve some resources for the learning plane, which confirms that the cost of learning is not negligible and needs to be accounted for when designing the resource allocation strategies, as we have done in our "Dynamic" scheme. The bar plots in Fig. 8a-d show that our scheme can outperform the static FDMA and TDMA resource allocation strategies, almost reaching the same performance as the ideal out-of-band system. The only cases with an appreciable performance gap between our scheme and the out-of-band system are env. 12 and env. 110: as we remarked above, these are the most challenging ones, with a total load close to or over 99% of the nominal link capacity. In these limited cases, any learning policy that requires resources for the training will unavoidably determine the violation of the QoS requirements for some users, which further highlights the importance of the cost of learning in the system design. Tuning $\rho(t)$ adaptively results in slightly higher performance in some cases, most appreciably in env. 110, but the difference is limited in most cases, as the performance gap with respect to the out-of-band ideal optimum is mostly due to the initial training and exploration phase in each environment.

The relative simplicity of the system model we considered makes it possible to analyze in depth the choices made by the different schemes. From the bar plots in Fig. 8e to Fig. 8l we can observe that the FDMA and TDMA schemes tend to drop or reject a significant number of packets in all environments, while the ideal and dynamic ones manage to limit the number of unserved packets for both slices. Interestingly, even the ideal scheme drops a significant number of packets from slice 1 in env. 0, but performance is still high. We can explain this by considering Fig. 9, which shows the empirical Cumulative Distribution Function (CDF) of the latency for packets in slice 2. Fig. 9a clearly shows that almost all packets have utility 1, i.e., are delivered before $T^{(2)}_{\text{soft}}$: in this case, all schemes tend to privilege slice 2, filling the queue in slice 1 more often. We should also consider that the learning agents start from scratch in environment 0, i.e., they have



(a) Env. 0 (b) Env. 12 (c) Env. 102 (d) Env. 110

FIGURE 9. Empirical CDF of the latency for slice 2 in the four selected environments.

no pre-trained weights to start from, and we should expect a relatively large number of mistakes.

We also observe that the ideal and dynamic schemes have matching latency profiles in env. 102, as Fig. 9c shows, while the FDMA and TDMA schemes tend to transmit more packets with a latency close to $T^{(2)}_{\max}$. In environments 12 and 110, shown in Fig. 9b and Fig. 9d, respectively, our scheme drops more packets than the ideal one, and has a higher overall latency, but still outperforms the static allocation schemes. Adaptively setting $\rho(t)$ can improve the performance of the latency-sensitive slice significantly in both of these environments, leading to the performance improvements mentioned in the previous paragraph. Interestingly, the two TDMA schemes tend to have better latency performance than the dynamic scheme in env. 12, but cannot improve the utility: the fraction of packets with latency higher than $T^{(2)}_{\text{soft}}$ is the same for all three schemes, and the two TDMA ones drop a large number of packets, causing a significant performance difference. In this case, serving most packets from slice 2 as soon as they arrive is not advantageous, as it leads to worse performance overall for TDMA.

VII. CONCLUSION AND FUTURE DIRECTION

In this work, we have designed a dynamic resource allocation policy, which can mediate between the learning and data planes, controlling the trade-off between the effectiveness and efficiency of DRL models. Unlike most works in the learning-based networking literature, we specifically consider the cost of learning, i.e., the resources required by the training process of a DRL agent, and show that our dynamic policy can outperform static schemes and, after a short transition phase, match the performance of an ideal system with an out-of-band learning plane. The adaptability of the scheme is demonstrated by applying it in a CL setting with environment changes, to which the dynamic scheme adapts extremely quickly.

Possible extensions of the work include the adaptation of the scheme to more complex scenarios, with a larger number of resources and slices and more stringent QoS requirements, as those for Ultra-Reliable Low-Latency Communications (URLLC). More advanced CL models could be implemented, with the goal of further improving the performance of the overall learning system. Furthermore, as mentioned before, the detection of context changes that trigger retraining of the network is another open question.

A critical challenge is to analyze the interplay between the cost of learning and active learning, which requires selecting the most valuable samples to be transmitted to accelerate the training, particularly when resources in the learning plane are scarce. Finally, of particular interest is the design of meta-learning schemes that can learn when the resource allocation scheme needs to be retrained, balancing the potential performance improvement that could be brought about by a retrained policy and the cost to learn it, relative to the expected system coherence time.

REFERENCES

[1] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, and Y. A. Zhang, "The roadmap to 6G: AI empowered wireless networks," *IEEE Commun. Mag.*, vol. 57, no. 8, pp. 84–90, Aug. 2019.

[2] M. Yang, Y. Li, D. Jin, L. Zeng, X. Wu, and A. V. Vasilakos, "Software-defined and virtualized future mobile and wireless networks: A survey," *Mobile Netw. Appl.*, vol. 20, no. 1, pp. 4–18, Feb. 2015.

[3] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2429–2453, 3rd Quart., 2018.

[4] N. C. Luong et al., "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3133–3174, 4th Quart., 2019.

[5] H. Sami, H. Otrok, J. Bentahar, and A. Mourad, "AI-based resource provisioning of IoE services in 6G: A deep reinforcement learning approach," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, pp. 3527–3540, Sep. 2021.

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[7] M. Delange et al., "A continual learning survey: Defying forgetting in classification tasks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 7, pp. 3366–3385, Jul. 2021.

[8] N. Shalavi, G. Perin, A. Zanella, and M. Rossi, "Energy efficient deployment and orchestration of computing resources at the network edge: A survey on algorithms, trends and open challenges," 2022, *arXiv:2209.14141*.

[9] I. Chih-Lin, "AI as an essential element of a green 6G," *IEEE Trans. Green Commun. Netw.*, vol. 5, no. 1, pp. 1–3, Mar. 2021.

[10] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[11] F. Mason, F. Chiariotti, and A. Zanella, "No free lunch: Balancing learning and exploitation at the network edge," in *Proc. IEEE Int. Conf. Commun.*, May 2022, pp. 631–636.

[12] F. Mason, G. Nencioni, and A. Zanella, "Using distributed reinforcement learning for resource orchestration in a network slicing scenario," *IEEE/ACM Trans. Netw.*, vol. 31, no. 1, pp. 88–102, Feb. 2023.
[13] S. Lahmer, F. Chiariotti, and A. Zanella, "The cost of learning: Efficiency vs. efficacy of learning-based RRM for 6G," in *Proc. IEEE Int. Conf. Commun.*, May 2023, pp. 5166–5172.
[14] X. Li, Y. Zhou, T. Wu, R. Socher, and C. Xiong, "Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3925–3934.
[15] A. Graves et al., "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, Oct. 2016.
[16] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne, "Experience replay for continual learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 350–360.
[17] D. Isele and A. Cosgun, "Selective experience replay for lifelong learning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, 2018, pp. 1–8.
[18] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 2935–2947, Dec. 2017.
[19] H. Ahn, S. Cha, D. Lee, and T. Moon, "Uncertainty-based continual learning with adaptive regularization," in *Proc. NIPS*, 2019, pp. 4392–4402.
[20] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, 2017, p. 3987.
[21] A. Mallya and S. Lazebnik, "PackNet: Adding multiple tasks to a single network by iterative pruning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7765–7773.
[22] J. Serra, D. Suris, M. Miron, and A. Karatzoglou, "Overcoming catastrophic forgetting with hard attention to the task," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 4548–4557.
[23] J. Schwarz et al., "Progress & compress: A scalable framework for continual learning," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 4528–4537.
[24] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 9, pp. 5149–5169, Sep. 2021.
[25] C. Finn, A. Rajeswaran, S. Kakade, and S. Levine, "Online meta-learning," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 1920–1930.
[26] O. Wichrowska et al., "Learned optimizers that scale and generalize," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, 2017, pp. 3751–3760.
[27] R. Houthooft et al., "Evolved policy gradients," in *Proc. 32nd Conf. Neural Inf. Process. Syst.*, 2018, pp. 5405–5414.
[28] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4780–4789.
[29] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, Aug. 2017, pp. 1126–1135.
[30] K. O. Stanleyand, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nat. Mach. Intell.*, vol. 1, pp. 24–35, Jan. 2019.
[31] J. Xu and Z. Zhu, "Reinforced continual learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 907–916.
[32] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 4080–4090.
[33] A. Antoniou, H. Edwards, and A. Storkey, "How to train your MAML," in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–11.
[34] M. Andrychowicz et al., "Learning to learn by gradient descent by gradient descent," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2016, pp. 3988–3996.
[35] A. Nagabandi et al., "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," in *Proc. 7th Int. Conf. Learn. Represent.*, 2018, pp. 1–17.
[36] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-reinforcement learning of structured exploration strategies," in *Proc. NIPS*, 2018, pp. 5307–5316.
[37] L. Yang and A. Shami, "A lightweight concept drift detection and adaptation framework for IoT data streams," *IEEE Internet Things Mag.*, vol. 4, no. 2, pp. 96–101, Jun. 2021.
[38] A. Bifet, "Adaptive learning and mining for data streams and frequent patterns," *ACM SIGKDD Explor. Newslett.*, vol. 11, no. 1, pp. 55–56, Nov. 2009.
[39] E. R. Faria, I. J. C. R. Gonçalves, A. C. P. L. F. de Carvalho, and J. Gama, "Novelty detection in data streams," *Artif. Intell. Rev.*, vol. 45, no. 2, pp. 235–269, Feb. 2016.
[40] V. Mnih, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

SEYYIDAHMED LAHMER (Student Member, IEEE) received the first bachelor's degree in computer science from the University of Chlef, Algeria, and the second bachelor's and master's degrees (Hons.) in computer science from Strasbourg University. He is currently pursuing the Ph.D. degree with the University of Padua. He is a Marie Curie Fellow with the University of Padua. His research interests include networked systems, machine learning, and computer architecture.

FEDERICO MASON (Member, IEEE) received the master's and Ph.D. degrees in telecommunication engineering from the University of Padua, Italy, in 2018 and 2023, respectively. In 2022, he was a Visiting Research Scholar with the Scripps Research Translational Institute, La Jolla, CA, USA. In 2023, he was a Post-Doctoral Researcher with the IRCCS Institute of Neurological Sciences of Bologna, Italy. Since October 2023, he has been an Assistant Professor with the Department of Information Engineering, University of Padua. His main research interests include the development of new computational techniques to process physiological signals and the analysis of the training costs in learning-based network optimization.

FEDERICO CHIARIOTTI (Member, IEEE) received the Ph.D. degree in information engineering from the University of Padua, Italy, in 2019. From 2020 to 2022, he was with Aalborg University, Denmark. He is currently an Assistant Professor with the Department of Information Engineering, University of Padua. He has published more than 80 peer-reviewed articles. His research interests include age of information, the use of machine learning in networks, and goal-oriented communications. He received the best paper awards in four conferences.

ANDREA ZANELLA (Senior Member, IEEE) received the Laurea and Ph.D. degrees in computer engineering from the University of Padova, Italy, in 1998 and 2001, respectively. He is currently a Full Professor with the Department of Information Engineering (DEI), University of Padua. He is one of the coordinators of the SIGnals and NETworking (SIGNET) Research Laboratory. His long-established research activities are in the fields of protocol design, optimization, and performance evaluation of wired and wireless networks.

Open Access funding provided by 'Università degli Studi di Padova' within the CRUI CARE Agreement